NTATION PAGE

Form Approved OMB No. 0704-0188

②

(Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED FINAL 15 Dec 90 - 14 Dec 92
4. TITLE AND SUBTITLE "STRATEGIC CONTROL OF REACTIVE BEHAVIOR IN INTELLIGENT AGENTS" (U)		5. FUNDING NUMBERS 62702F 5581/00(RADC)
6. AUTHOR(S) Barbara Hayes-Roth, Lee Brownston & Anne Collinot		
ORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ford Univrsity uter Science ledge Systems Laboratory ford CA 94305		8. PERFORMING ORGANIZATION REPORT NUMBER AFOSR-TR-
ONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES) SR/NM Duncan Ave, Suite B115 ling AFB DC 00001		10. SPONSORING MONITORING AGENCY REPORT NUMBER AFOSR-91-0131
SUPPLEMENTARY NOTES		
a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; Distribution unlimited		12b. DISTRIBUTION CODE UL

A series of experiments was conducted to evaluate the real-time performance of a proposed agent architecture. The architecture is a blackboard architecture whose key features include: distribution of perception, action, and cognition among parallel processes; limited-capacity I/O buffers with best-first retrieval and worst-first overflow; dynamic control planning; dynamic focus of attention; and a satisficing algorithm for the execution cycle. The experiments focus on the architecture's satisficing algorithm for the execution cycle, which is the unit-process of all reasoning and critical to real-time performance. The experiments focus on the architecture's satisficing algorithm for the execution cycle, which is the unit-process of all reasoning and critical to real-time performance. The execution cycle has three steps: (1) notice possible operations; (2) choose the best operation with respect to the current control plan; and (3) execute the chosen operation. Because the executed operations can change the agent's control plan, this cycle allow the agent to dynamically construct and modify plans that control its own behavior. The problem with this cycle in a real-time context is the unbounded time associated with step 1. The satisficing solution is to use the control plan of step 2 to order the noticing of possible actions in step 1 and to interrupt step 1 whenever the agent notices an operation that is "good enough" or a deadline occurs. The general hypothesis is that, with high quality (specific and correct) control plans, an agent will be able to identify high-priority actions very quickly, well within deadline,. Thus, deadlines serve as a rarely used "last line of defense" against unbounded cycle time.

14. SUBJECT TERMS			15. NUMBER OF PAGES ~~31~~ 17
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89) Prescribed by ANSI Std 239-18 298-102 final report!

Knowledge Systems Laboratory
Report No. KSL 93-18

February 1993

Strategic Control of Reactive Behavior in Intelligent Agents

by

Barbara Hayes-Roth
Lee Brownston
Anne Collinot

KNOWLEDGE SYSTEMS LABORATORY
Department of Computer Science
Stanford University
Stanford, California 94305

Strategic Control of Reactive Behavior in Intelligent Agents [1]

Barbara Hayes-Roth, Lee Brownston, and Anne Collinot

Stanford University
University of Paris VI

October, 1992

Abstract: An intelligent agent must often choose among several competing actions. To act effectively in real time, it must both generate strategic plans and respond to external events. Constructing plans completely in advance lacks flexibility, while purely reactive systems can lack generality. We present an algorithm which dynamically constructs a control plan and yet uses this plan in a reactive fashion when response time is critical. This Satisficing algorithm uses the control plan to process both events and known actions in a most-promising-first order, terminatig its search whenever it has either encountered a potential action that exceeds a declared threshold, reached a temporal deadline, or exhaused all known actions, whichever comes first. Experiemental results are presented to demonstrate that the satisficing algorithm has the expected and desired characteristics.

Area: Architectures and languages for AI

[1] Research and preparation of this paper were supported by NASA contract NAG 2-581 under DARPA Order 6822 and AFOSR grant AFOSR-91-0131.

Declaration: This paper is neither currently under review for a journal or another conference, nor will it be submitted during IJCAI's review period.

1. Controlling Behavior in Intelligent Agents

Consider an agent whose job is to monitor a ventilator-assisted patient in the intensive care unit. The agent is busy making a plan for gradually reducing the amount of assistance provided by the ventilator and eventually weaning the patient. In the midst of its planning, the agent perceives an abnormally high value of the peak inspiratory pressure ("PIP") required to inflate the patient's lungs. At that point, several actions are available to the agent, including: (a) continue planning; (b) report the observed high PIP to a physician; (c) begin to diagnose the problem underlying the high PIP; (d) recommend an action to give the patient more oxygen; or (e) look for interesting patterns in other perceived patient data. How does the agent decide which action to take?

As this simple example illustrates, any non-trivial agent faces a challenging *control problem.* The agent is bombarded by a continuing stream of events, some generated internally and some externally. Each event may suggest many possible actions, some of which would be executed internally and some externally. But the agent cannot execute them all. How does it choose among many possible actions at each point in time?

There are two main approaches to control in the AI literature. In the "classical planning model" [Fikes and Nilsson, 1971], an agent reasons about actions, events, and goals ahead of time to construct a plan of specific actions that will achieve its goal. At run-time, it executes the planned actions. When the agent's assumptions are correct, this approach produces globally effective behavior and minimizes the cost of run-time perception and action selection. However, detailed planning is computationally expensive [Chapman 87] and plans are vulnerable to run-time events. The agent cannot respond to unanticipated demands and opportunities for action without an expensive, time-consuming replanning process. In the "reactive model" [Agre and Chapman 87; Kaelbling 87], an agent constructs, or is programmed with, a compiled network of goal-specific perception-action rules. At run-time, it uses perceptual events to select actions for execution. When the number and rates of perceived variables are fixed, this approach allows the agent to respond in bounded time to a range of possible events. However, it is difficult to construct complete networks for complex task environments [Ginsberg 89], it is difficult to produce globally-effective behavior from locally effective actions, and reactive networks are useless when goals change [Maes 90], knowledge grows, or unanticipated events occur. Thus, although planning and reactive models work well for particular classes of problems, neither of them solves the control problem for intelligent agents.

As we have discussed elsewhere [Hayes-Roth and Hayes-Roth79; Hayes-Roth 85; Hayes-Roth 90; Hayes-Roth, et al., 92], a control architecture for intelligent agents must integrate the strengths of both planning and reactive models, while avoiding their limitations.

An intelligent agent should make and follow strategic plans. Given some global objectives, it should decide which short-term goals to pursue and how to pursue them — based on its run-time circumstances. Only in rare cases will the agent need (or be able) to make a complete and detailed plan of the sort produced by classical planners. More often, a rough "outline" of the intended course of action will suffice. For example, our monitoring agent might observe some unexpected patient data and set a short-term goal to diagnose the underlying problem. If the observation is critical (e.g., high PIP), the agent might plan a specific series of tests to confirm or disconfirm life-threatening conditions (e.g., a hole in the lung, called a "pneumothorax") as soon as possible. If the observation is not critical (e.g., low temperature), the agent might decide only to track the observed variable and related variables a little more closely to see what develops. If the agent's circumstances entail multiple goals (e.g., simultaneous observations of high PIP and low temperature), the agent can decide how to coordinate its efforts to achieve them, for example by interleaving their component actions, by performing actions that address both of them, or by postponing or permanently ignoring one of them.

An intelligent agent should react to run-time events. It can only execute actions that are possible at run time and it can only evaluate their utility in their run-time context. So there usually is no need and no payoff for the agent to try to anticipate each and every action it will perform. Instead, possible actions become apparent as the agent goes about its business. It should execute those that advance its current plans, but it also may execute unplanned actions. For example, in trying to predict the consequences of a patient's unexpectedly low temperature, a monitoring agent can estimate the patient's blood gases either by: (a) inspecting continuously sensed data — if an oximeter is in place; or (b) requesting a laboratory test and waiting twenty minutes for the results — if a nurse is available to draw blood. If the monitoring agent has no immediate goals, it might nonetheless perform actions to track and interpret key patient variables--a potentially useful expenditure of slack computational resources.

To address these requirements, our agent architecture provides a uniform "generate-and-test" mechanism to control all behavior: (1) The agent reactively generates possible actions based on run-time events. (2) The agent tests possible actions against its current strategic plan and executes the best one. Because the model does not distinguish internal from external behavior, the agent sometimes generates, tests, and executes actions that change its strategic plan, thereby changing the criteria it uses to test future actions. Thus, the agent architecture uses reactively-constructed strategic plans to control the selection and execution of reactively-generated actions.

In addition to integrating strategic and reactive behaviors, our control architecture has other desirable properties. It simplifies the knowledge acquisition problem because an agent need know only what events enable each of its actions, not every combination of goals and events that make each action the best alternative. It simplifies planning because an agent can plan classes of actions to execute during time intervals, rather than planning individual actions to execute at time points. It increases versatility and extensibility because the agent can use different combinations of actions under different strategic plans to achieve different goals; it can acquire new actions, plans, and goals independently. It permits explanation of behavior in terms of the strategic decisions that actually produced it. Finally, it introduces the first real possibility of executing possible actions that have not been planned at all, either because the agent has adopted the "universal plan" or because it chooses to act in any way possible when no currently available action matches its plan.

We have experimentally demonstrated and evaluated these properties of the proposed control architecture in two classes of agents. In a class of design agents [Hayes-Roth, et al., 86; Garvey, et al., 87; Johnson, et al., 87; Tommelein, et al., 91], the need for reactivity arises primarily from uncertainty in the effects of the agent's own actions, with some additional uncertainty from a human user's discretionary interventions in the problem-solving process. The need for strategic control arises from differences in problem-solving efficiency for different sequences of actions. In a class of monitoring and control agents [Hayes-Roth, et al., 89; Hayes-Roth, et al., 92; Murdock and Hayes-Roth 91], there is considerable additional uncertainty in the exogenously-determined behavior of the monitored system, as well as in its responses to the agent's actions.

And the agent can choose among a larger repertoire of possible actions, influencing not only efficiency, but effectiveness, timeliness, completeness, and other properties of a multivariate utility function.

For monitoring agents and other agents operating in dynamic environments, the simple form of our control mechanism has a serious weakness. If an agent reactively generates <u>all</u> possible actions, the computation required to complete a "cycle," its basic unit of behavior, is unbounded with respect to key features of its task environment: event rate and number of known actions. If *n* is the number of observed events and *k* is the number of known actions, in the worst case, the time to generate all possible operations is *O(nk)*. Translating into real time, an agent's rate of operation slows down as its environment increases in volatility and as its knowledge increases. This is nonintuitive for a so-called "intelligent" agent and catastrophic for an agent that must meet real-time constraints. The problem is not solved by introducing multiple processors because any agent will have limited resources and the possibility of situations that exceed its resources.

An intelligent agent must reconcile limited computational resources with an effectively unbounded task environment.

At a minimum, an agent must maintain a stable "rate of thought," which we operationalize as a bounded cycle time that is immune to event rate and number of known actions. Thus, the agent could guarantee a maximum latency for simple reactions (e.g., to signal an alarm after perceiving critical data), reliably estimate its latency for multi-action behaviors (e.g., to diagnose an observed problem), and plan multi-action behaviors that meet deadlines (e.g., to diagnose high PIP, with a four-minute deadline, versus low temperature, with no particular deadline). Because absolute cycle time reflects implementation artifacts, it is less important than bounded cycle time and protection from complexity. If we can meet these objectives, absolute cycle time can be speeded up by a constant factor with conventional techniques.

Ideally, an agent should be able to "think fast" or "think carefully," as appropriate, which we operationalize as a deliberate modulation of cycle time within some range. Thus, the agent could deliberately improve the latency of its responses to urgent events (e.g., to act immediately to avert a life-threatening condition) or, conversely, improve the expected quality of its responses to other events (e.g., to identify the most specific diagnosis and the optimal response to a complex condition). Obviously, the agent could make these adaptations only by trading speed for quality in the task at hand or by compromising other aspects of its behavior, such as its (non)performance of competing tasks.

2. A Satisficing Algorithm for Real-Time Control

To address these requirements, we specialize our architecture's "generate-and-test" mechanism with a parameterized *satisficing algorithm*, as illustrated in Figure 1.

First, the satisficing algorithm bounds the number of observed events and known actions the agent considers on each cycle. A fixed-capacity event buffer bounds the number of observed events to a constant, n_b. The buffer overflows worst-first based on event ratings against the control plan. An event discrimination tree bounds the number of actions considered for a given event to a variable, k_b, the number of "relevant" actions in the tree. From these, it further prunes actions worst-first, based on rating against the control plan. Thus, cycle time is strictly bounded by the product $n_b k_b$.

Second, the satisficing algorithm orders the generation of possible actions on each cycle by moving some of the "test" of our generate-and-test mechanism into the "generator." Ratings against the control plan are used to order the selection of observed events from the buffer and, for a given event, to order the selection of retrieved relevant actions. For each event-action pair whose additional conditions are true, a possible action is created, rated, and placed on an agenda that has fixed capacity (a stable parameter) and worst-first overflow. Other things equal, a possible action based on a highly-rated observed event or a highly-rated retrieved action will itself have a high rating. However, the possible actions cannot be generated strictly best first for several reasons: possible actions may be based on a highly-rated event or action, but not both; possible actions may be based on continuous, rather than binary, ratings of events and actions; possible actions may have different ratings against control plans for competing tasks. Thus, possible actions are generated roughly best first. The control plans that determine the degree of best-first generation are parameters set dynamically by the agent.

Third, the satisficing algorithm terminates generation of possible actions and executes the best one generated so far under any of three conditions. (1) The algorithm terminates whenever a possible action is generated that exceeds a threshold rating with respect to the current control plan. In that case, it executes an action that is "good enough" as soon as possible. (2) The algorithm terminates whenever a cycle deadline occurs. In that case, it executes the best available action when necessary. (3) The algorithm terminates whenever all observed events and retrieved actions have been considered. In that case, it executes the best possible action under the circumstances. The two interrupt conditions, good-enough threshold and cycle deadline, are parameters set dynamically by the agent.

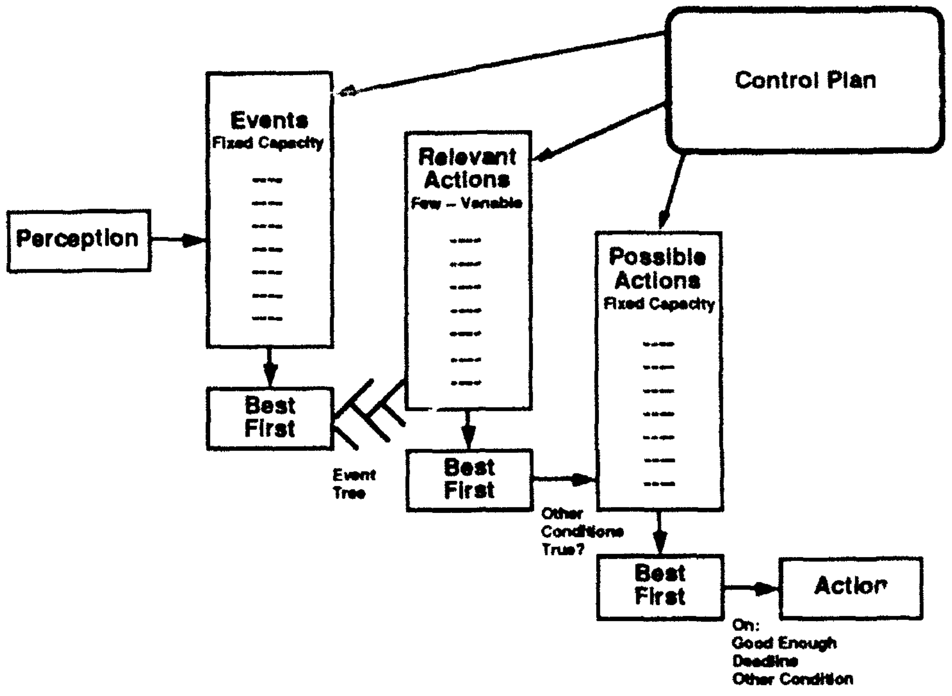

Figure 1. Using control plans to generate possible actions roughly best first.

3. Bounding Cycle Time in an Unbounded Task Environment

Our first prediction is that, with appropriate paran..terization of the satisficing algorithm, an agent can bound its cycle time, despite increases in the values of key variables of its task environment: observed event rate and number of known actions. Of course, the agent can bound cycle time directly, by adopting a deadline. But the shorter the deadline, the greater the risk of executing incorrect actions on some cycles. What we want are correct actions on short, bounded-time cycles. The solution we propose is to speed up average cycle time within a conservative deadline by introducing an interrupt that produces a "good enough" action prior to deadline whenever possible.

In the present experiment, we define "good enough" intuitively as a high rating on both event and action against a very important control decision. Operationally, we define it as a high absolute threshold for a possible action's *weighted rating* against a single control decision: $WR = ((ER + AR) / 2) * I$, where ER is the 0-1.0 rating of the possible action's event against the event specified in the control decision, AR is the 0-1.0 rating of the possible action's action against the action in the control decision, and I is the importance of the control decision. For example, assume that Guardian, our prototype patient-monitoring system, has made two concurrent control decisions: a very important one, "Respond-to critical data;" and a less important one, "Interpret new data." Assume that two events co-occur: "observed high PIP" is a critical event; and "observed low temperature" is a non-critical event. Finally, assume that for each of these events Guardian knows three relevant actions: "react-to" is a member of the action class "respond-to;" "explain" and "predict-effects-of" are members of the action class "interpret." Six possible actions could be generated in this situation, with different ratings weighted against the two control decisions, as illustrated in Figure 2. Only one of them, "React-to high PIP," gets high ratings on both event and action against the very important control decision, "Respond-to critical events." Therefore, only "React-to high PIP" would be "good enough" to execute immediately after generation and prior to deadline. With this definition of "good enough" and a conservative deadline, Guardian cycles never run until deadline during the experimental scenario discussed below; cycles always are interrupted by generation of a possible action that is good enough.

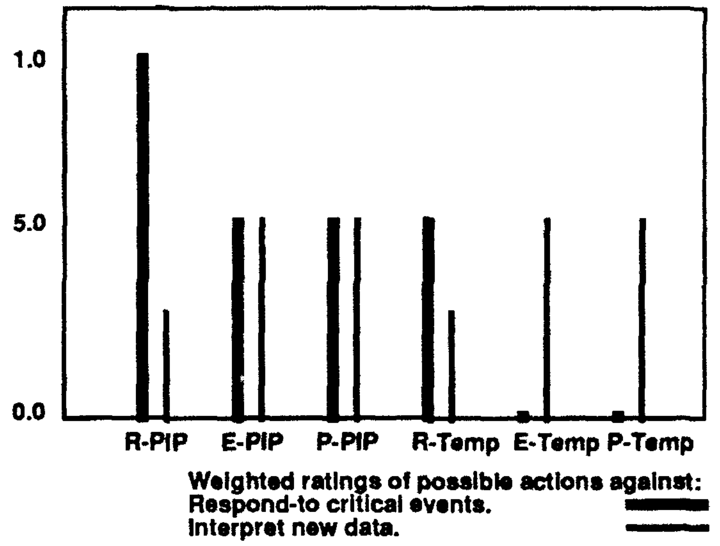

Figure 2. Weighted ratings of six possible actions against two concurrent control plans.

To test the predictions, we measured Guardian's cycle time during a carefully controlled segment of a longer monitoring scenario. The segment begins when the patient simulator produces an abnormally high value of the patient's peak inspiratory pressure; it ends 23 cycles later, when Guardian recommends the corrective therapy of inserting a chest tube. During the entire segment, Guardian executes these actions:

Make a control decision:

"React-to high PIP (peak inspiratory pressure)."

Perform a series of diagnostic reasoning actions.

Diagnose problem class: "Hypoxia."

Recommend useful interim action:

"Increase MV (minute ventilation) to insure adequate oxygen."

Perform a series of diagnostic reasoning actions.

Diagnose underlying problem: "Pneumothorax (hole in the lung)."

Recommend corrective action:

"Insert a chest tube to release accumulated air from the lungs."

To control Guardian's internal state at the time measurements were made, we programmed our patient simulator to produce the first high PIP event and begin the experimental segment about 55-60 cycles into the larger monitoring scenario. More specifically, it occurs immediately after Guardian sets up its monitoring parameters, creates its default control plan, adjusts its sensed data rates, and begins classifying observed patient data, but before it performs any other reasoning tasks. Thus, in all conditions, the experimental segment begins immediately after Guardian reaches the same internal steady-state (e.g., current control plan, ongoing tasks, memory load, memory contents). Otherwise, differences in internal state could produce artifacts in our measurements.

We manipulated three experimental variables, the event rate during the experimental segment, the number of different actions Guardian knows, and the two together. In the "base" condition, approximately 280 events occur (an average of 12 per cycle) and Guardian knows 51 different actions. To produce the other experimental conditions, we replicated the base condition events and actions 2, 4, and 8 additional times, separately and together. Thus, in the worst (that is, most heavily burdened) condition, covariation at 8 times base rate, approprimately 2240 events occur (an average of 97 per cycle) and Guardian knows 408 different actions.

We measured cycle time, excluding action execution time, which is not covered by our predictions. To reduce "noise" in the data, we repeated all conditions three times. Thus, the average cycle time for each condition represents 69 observations (3 replications on each of 23 cycles); the data are highly reliable.

For comparison, we performed exactly the same manipulations and measurements with an "exhaustive" algorithm. It uses the same event discrimination tree used by the satisficing algorithm to bound the number of relevant actions considered for each event, but otherwise tries to generate possible actions for every known action that is relevant to every observed event.

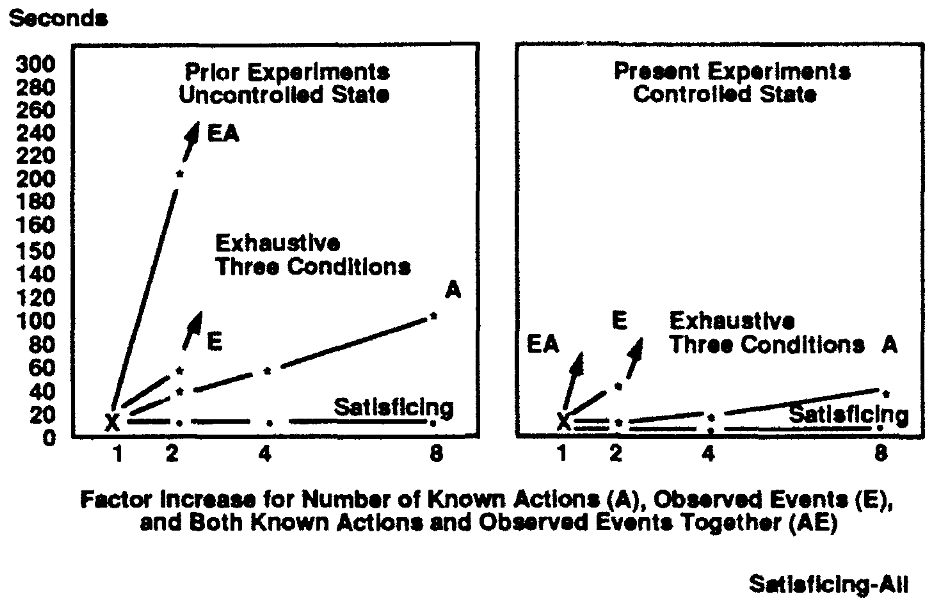

Figure 3. Effects of complexity factors on cycle time for the satisficing and exhaustive algorithms.

Results appear in the right panel of Figure 3. As predicted, the satisficing algorithm produces effectively constant cycle time, despite increases in the number of observed events, the number of known actions, or the two variables together. The three lines representing these three independent variables are effectively identical and are superimposed in Figure 3. The satisficing algorithm is insensitive to event rate because, on each cycle it very quickly identifies the best available event and uses it to try to generate an executable instance of the best of the relevant actions; the resulting possible action typically is good enough to execute immediately. Other less important events simply accumulate and overflow the event buffer, worst first. The satisficing algorithm is insensitive to number of known actions because, for a given event, it uses the event tree to bound the number of relevant actions it considers and very quickly identifies the best of those; again, the resulting possible action typically is good enough to execute immediately. Other less important actions are never considered. Increasing event rate and number of known actions together poses no new problems for the satisficing algorithm.

By contrast, with the exhaustive algorithm, cycle time appears to be a linear function of number of known actions and a positively accelerated function of event rate. In fact, cycle time grows so fast with increases in event rate and the covariation that it was impractical to complete the experiments in those conditions. This is because the exhaustive cycle considers every observed event and for each one, every relevant action. Increasing the number of known actions is not as bad because the exhaustive cycle uses the event discrimination tree to bound the number of relevant actions it considers for each event. In our experiments, there are only 2-3 actions relevant to each observed event in the base condition and, therefore, only 16 or 24 in the extreme conditions where the base number is increased by a factor of 8. The linear effect remains because the exhaustive cycle must consider all of the relevant actions. In agents that know significantly more actions or actions with weaker enabling conditions, the event tree alone might not bound relevant actions so effectively and the effect of number of known actions would increase in magnitude.

These results replicate the qualitative findings of a prior experiment [Hayes-Roth and Collinot, 1993], shown in the left panel of Figure 3. However, in contrast to our careful control of Guardian's state in the present experiment, the prior experiment confounded state with manipulations of the independent variables. In that experiment, we programmed our patient simulator so that the high PIP occurred after a fixed real-time delay, during which Guardian perceived and responded to other events (e.g., abnormal patient data or questions from a user). Since the number of cycles Guardian completes during any given real-time delay is inversely related to its cycle time, Guardian began the experimental segment after a different number of cycles and, therefore, in a different internal state in different experimental conditions. Because it is very difficult to characterize all of Guardian's state variables and very difficult to predict their effects on cycle time in different experimental conditions, the methodology of the prior experiment casts some doubt on the interpretation of its results. Thus, the present experiment provides an essential replication of the earlier results under better controlled conditions.

Comparing results of the two experiments, the only obvious difference is that cycle time is consistently shorter in the present experiment than in the prior experiment. This is probably not an effect of controlling state so much as an effect of making our measurements early in the larger monitoring scenario. In the prior experiment, Guardian does more reasoning before the experimental segment begins, producing more information in memory, which generally slows its performance. This phenomenon is independent of our predictions and experimental manipulations.

The important finding is that the two experiments produced qualitatively similar results, confirming the satisficing algorithm's ability to bound cycle time under a variety of conditions.

We have confirmed the sufficiency of the satisficing algorithm to bound cycle time. But are all of the algorithm's key features necessary?

To answer this question, we performed the same experiment with two variations on the satisficing algorithm. Table 1 compares these algorithms with the satisficing and exhaustive algorithms. The satisficing algorithm has five key features: bounding of the number of (1) events and (2) actions considered during generation of possible actions; best-first ordering of the retrieval of (3) events and (4) actions for consideration; and (5) interruption of possible action generation to execute an action that is good enough. The unbounded-events algorithm eliminates bounding of the number of events considered. The interrupt-only algorithm also eliminates the best-first ordering of event and action retrieval. The exhaustive algorithm also eliminates the "good enough" interrupt.

Table 1. Four Generate-and-Test Algorithms

Key Features of the Satisficing Algorithm

Algorithm	**Bounded Variables**		**Ordered Retrieval**		**Interrupt**
	Events	**Actions**	**Events**	**Actions**	
Satisficing	x	x	x	x	x
Unbounded-events	-	+	x	x	x
Interrupt-Only	-	+	-	-	x
Exhaustive	-	+	-	-	-

Figure 4 shows the results for the Satisficing, Interrupt-Only, and Exhaustive algorithms. With the alternative algorithms, cycle time increases linearly with number of known actions. It increases more than linearly with event rate — so much that it became impractical to measure it at the higher event rates. Cycle time increases even faster with the covariation of event rate and number of known actions, so we were unable to collect data for the alternative algorithms in most of those conditions. The results for the Unbounded-Events algorithm, not shown in Figure 4 so as to reduce clutter, showed that cycle time grew negligibly as a result of number of known actions, but grew explosively when event rate increased.

Note that our experiment is designed so that Guardian will execute the correct sequence of actions described above in all conditions, regardless of algorithm, if given enough time and computational resources. However, even without network crashes and even assuming a generous constant factor speed-up, the alternative algorithms would execute the correct actions much too late to produce their intended effects in a real-time environment. Only the satisficing algorithm produces correct actions with constant cycle time, despite increases in event rate and number of known actions. These results argue for the necessity, as well as the sufficiency, of all key features of the satisficing algorithm to bound cycle time in unbounded task environments.

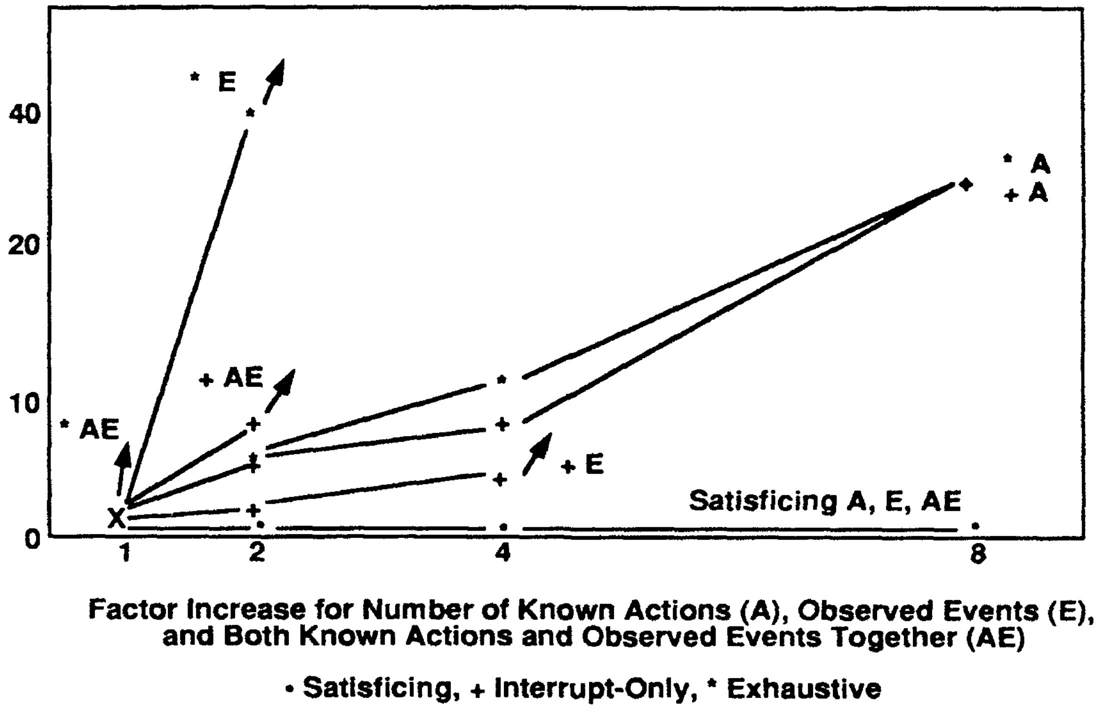

Figure 4. Comparison of cycle times, with variation in event rate and number of known actions, for four algorithms.

The observed results depend on an appropriate parameterization of the satisficing algorithm, in particular an appropriate definition of "good enough," given the task environment. The absolute threshold we used to define "good enough" is appropriate for agents whose task environments require them to "think fast" in easily recognizable emergency situations. Alternatively, an agent might wish to "think fast" all of the time, even in non-emergency situations--in which case, it might be better to define "good enough" relative to the most important active control decision, rather than in terms of an absolute threshold. We return to this point below. For now, we note that the definition of "good enough" is a parameter that the agent designer or the agent itself can set and modify as appropriate for a dynamic task environment.

4. Modulating Cycle Time to Control the Speed of Thought

Our second prediction is that, with an appropriate parameterization of the satisficing algorithm, an agent can modulate its cycle time to "think fast" or "think carefully," at its own discretion. Again, the agent can modulate its cycle time directly by adopting different deadlines in different situations. However, as discussed above, deadlines are a crude instrument for controlling cycle time without unduly jeopardizing correctness. What we want are cycle time modulations that promote correctness. The solution we propose is to modulate cycle time indirectly by manipulating the discriminative power of control plans--intuitively, the degree to which a control plan discriminates a small number of possible actions that are "good enough" from a much larger number of possible actions that are not. A very discriminative control decision enables an agent to generate and execute an action that is good enough (or determine that no good enough action can be generated) very quickly. A less discriminative control decision requires the agent to generate a larger number of sub-threshold possible actions before generating one that is good enough (or determining that none can be generated).

In the present experiments, we operationalize the discriminative power of a control decision as: $DP = 1 - ((HE/OE + HA/KA) / 2)$, where HE is the number of observable events to which the control decision would give its highest possible event rating, OE is the number of observable event types, HA is the number of known actions to which the control decision would give its highest possible action rating, and KA is the number of known actions.

For example, Guardian might make one of these alternative control decisions: "React to high PIP," "Respond to important patient data," or "Reason about data." Assuming that observed events randomly sample the distribution of observable events and that relevant actions are normally distributed among observable events, these three decisions give the theoretical distributions of ratings illustrated in Figure 5. "Reason about data" is the least discriminative of the three decisions; it gives its highest possible rating to all possible actions. In the worst case. Guardian might have to try all combinations of observed events and relevant actions before generating a possible action that is good enough. "Respond to important patient data" is moderately discriminative; it gives normally distributed ratings with a mean ratings around 50. More importantly, it gives its highest rating to only about 20% of the possible combinations of observed events and relevant actions. Thus, Guardian can use it to bound the maximum number of observed events and relevant actions it must consider before generating a possible action that is good enough. "React to high PIP" is the most discrimintative of the three control decisions; it gives its highest possible rating to very few possible actions and its lowest possible rating to most possible actions. Guardian can use it to severely bound the number of observed events and relevant actions it must consider before generating a possible action that is good enough. Of course, these simplifying assumptions do not hold in any real agent and so we examine the predictions experimentally.

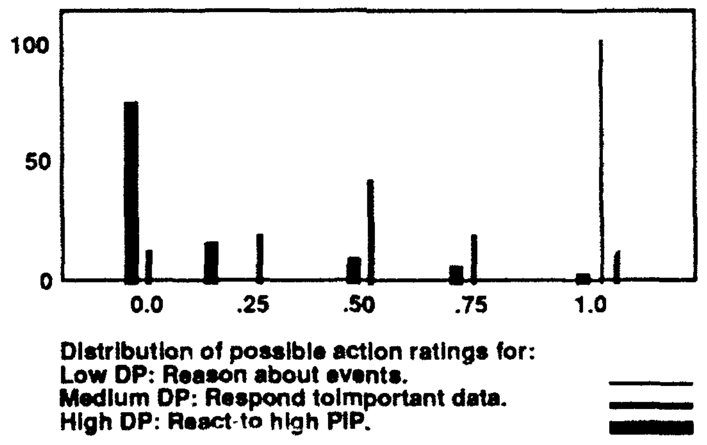

Figure 5. Theoretical distributions of ratings given to possible actions generated from observable events and known actions with respect to control plans of different discriminative power.

To test the predictions, we measured Guardian's cycle time during the same carefully controlled segment of the longer monitoring scenario described above, in the 4 times base rate condition for both observed events and known actions. We manipulated three experimental variables with respect to Guardian's control plan: discriminative power of the event description, of the action description, and of the two together. In a representative run of the experimental segment, event descriptions of low, medium, and high discriminative power gave their highest possible rating to 61, 100, and 144 of 283 observed events. Action descriptions of low, medium, and high discriminative power gave their highest possible ratings to x, 1.5x, and 2x of 408 known actions.

The results confirm the predictions with respect to control-plan event specificity. The cycle time was 2048 msec when the control plan event specificity was highest; 2175 msec for the medium level of control plan specificity; and 2444 msec for the least-specific control plan. The results, however, showed no systematic effect of control-plan action specificity, with the high-specificity condition having a cycle time only 34 msec different from that of the low-specificity condition — a difference of only about 1.5%.

5. Related Research

Although a number of researchers are studying ways to integrate planning and reactivity [Alterman 88; Drummond and Bresina 90; Durfee and Lesser 88; Georgeff and Lansky 87; Godefroid and Kabanza

91; Mitchell 90], the proposed satisficing algorithm is most similar to Maes's spreading activation mechanism for action selection. Basically, she defines a network of actions (which she calls "competence modules"), each of which is related to the environment by a *condition-list*, to global goals by its *add-list* and *delete-list*, and to other actions by *successor*, *predecessor*, and *conflictor* links. Actions receive activation when their conditions, add-list goals, predecessors, or successors are activated. They receive inhibition when their conflictors are activated or their delete-list goals are protected. Activation and inhibition continuously flow through the network, changing the *activation-levels* of the individual actions. An action is executable when all elements of its condition-list are true; an executable action is selected for execution when its activation-level reaches a certain threshold.

Despite their use of very different mchanisms, Maes's approach and ours share several key features:

- the potential to execute any known action at any time;
- the distinction between actions that are executable and those that are selected for execution;
- the use of goals and pre-conditions together to generate potential actions;
- the ability to operate under different goals without changing the basic mechanism.

We have a few quibbles with Maes's approach model (e.g., virtual sensors finesse an important part of the problem). However, overall, we believe the two approaches have complementary strengths. Maes's spreading-activation mechanism is more powerful and efficient than our symbolic mechanism, especially for more complex control plans than the simple event-action descriptions we used in the present experiments. In fact, even before reading Maes's paper, we had deveɔped a programmable spreading-activation mechanism for another purpose [Wolverton and Hayes-Roth 92] and considered using it in our generator. Perhaps now we will apply it to plan-constrained generation of possible actions. On the other hand, our mechanism offers a more powerful approach to goal-directed planning and control of action, especially for autonomous agents operating in more complex task environments. It would be interesting to integrate the strengths of both approaches in a single model.

6. Conclusions

The Satisficing algorithm is more complex than the Exhaustive algorithm and requires considerably more computation: maintaining the events in sorted order; checking for buffer overflow; carrying over from one cycle to the next the events and actions that remained unprocessed upon termination with a deadline or good-enough interrupt; and the execution of repeated tests for termination of search within a loop. What is gained is a control regime that is sensitive to external information while following a global plan, and that is apparently able to focus its processing, during critical situations, to such a high degree that it becomes immune to the effects of very heavy information loads.

The Satisficing algorithm successfully exploits the control plan not only to choose one of several possible actions, but also to prune its search for such possible actions. As evidence, we presented evidence that weakening the specificity of the control plan slows diminishes the efficiency of the search for possible actions. Furthermore, each component of the satisficing algorithm contributes its share. The results reported in Figure 4 and the accompanying text show that, when the optimizations of the Satisficing algorithm are successively relaxed, the algorithm dramatically loses its immunity to information load, quickly becoming overwhelmed when the number of known actions and, even more strikingly, the rate of observation events increase. The various components of the Satisficing algorithm, do not combine additively but rather act together synergistically to manage complexity, lending compelling evidence that this algorithm embodies is a principled as well as effective approach to the tradeoff between sensitivity to external data and goal-driven planning.

References

Agre, P. E. and Chapman, D. (1987) Pengi: an implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 268-272.

Alterman, R. Adaptive planning. *Cognitive Science*, 12(3), 393-421.

Chapman, D. (1987) Planning for conjunctive goals. *Artificial Intelligence*, 32(3), 333 - 378.

Drummond, M. and Bresina, J. (1990) Anytime sythentic projection: Maximizing the probability of goal satisfaction. *Proceedings of the Eighth National Conference on Artificial Intelligence*, 138-144.

Durfee, E. H. and Lesser, V. R. (1988) Predictability versus responsiveness: Coordinating problem solvers in dynamic domains. *Proceedings of the Seventh National Conference on Artificial Intelligence*, 66-71.

Fikes, R. E. and Nilsson, N. J. (1971) STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 198-208.

Garvey, A., Cornelius, C., and Hayes-Roth, B. (1987) Computational costs versus benefits of control reasoning. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 110-115 .

Georgeff, M. P. and Lansky, A. L. (1987) Preactive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 677-682.

Ginsberg, M. L. (1989) Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4), 40-44.

Godefroid, P. and Kabanza, F. (1991) An efficient reactive planner for synthesizing reactive plans. *Proceedings of the Ninth National Conference on Artificial Intelligence*, 640-645.

Hayes-Roth, B. (1985) A blackboard architecture for control. *Artificial Intelligence*, 26(3), 251-321.

Hayes-Roth (1990) Architectural foundations for real-time performance in intelligent agents. *Journal of Real-Time Systems*, 2(1/2), 99-125.

Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A., and Seiver, A. (1992) Guardian: A prototype intelligent agent for intensive-care monitoring. *Journal of Artificial Intelligence in Medicine*, 4, 165-185.

Hayes-Roth, B. (1992) Opportunistic control of action in intelligent agents. *IEEE Transactions on Systems, Man, and Cybernetics* (in press).

Hayes-Roth, B., Buchanan, B. G., Lichtarge, O., Hewett, M., Altman, R., Brinkley, J., Cornelius, C., Duncan, B., and Jardetzky, O. (1986)

Protean: Deriving protein structure from constraints. *Proceedings of the Fitth National Conference on Artificial Intelligence*, 904-909.

Hayes-Roth, B. and Collinot, A. (1993) A satisficing cycle for real-time reasoning in intelligent agents. *International Journal of Expert Systems and Applications* (in press)

Hayes-Roth, B. and Hayes-Roth, F. (1979) A cognitive model of planning. *Cognitive Science*, 3, 275-310.

Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S., and Cammarata, S. (1979) Modelling planning as an incremental, opportunistic process. *Proceedings of the Sixth International Joint Conference on Artifical Intelligence*, 375-383.

Hayes-Roth, B., Johnson, M. V., Garvey, A., and Hewett, M. (1986) Applications of BB1 to arrangement-assembly tasks. *Journal of Artificial Intelligence in Engineering*, .

Hayes-Roth, B., Washington, R., Hewett, R., Hewett, M., and Seiver, A. (1989) Intelligent monitoring and control. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 243-249.

Johnson, M. V., Jr., and Hayes-Roth, B. (1987) Integrating diverse reasoning methods in BB1. *Proceedings of the National Conference on Artificial intelligence*, 30-35.

Kaebling, L. (1987) An architecture for intelligent reactive systems. *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Los Altos, CA: Morgan Kaufmann.

Maes (1990) Situated agents can have goals. *Robotics and Autonomous Systems*, 6, 49-70.

Mitchell, T. M. (1990) Becoming increasingly reactive. *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1051-1058.

Murdock, J. and Hayes-Roth, B. (1991) Intelligent monitoring of semiconductor manufacturing. *IEEE Expert*, 6(), 19-31.

Tommelein, I., Levitt, R., Hayes-Roth, B, and Confrey, A. (1991) SightPlan experiments: Alternate strategies for site layout design. *Journal of Computing in Civil Engineering,* 5, 42-63. LSB ?

Wolverton, M. J. and Hayes-Roth, B. (1992) Using controlled knowledge search to retrieve cross-contextual analogies. Stanford University, Knowledge Systems Laboratory, Report KSL-92-72..